Receive Descriptor Recycling for Small Packet High Speed Ethernet Traffic

***Abstract*— This paper presents the concept of Receive Descriptor Recycling to significantly reduce the performance drop associated with small packet Gigabit Ethernet traffic. Since limits and trade-offs are inherent when optimising for small packet traffic, all important aspects in this context are covered. Low-level measurements were performed at the CERN LHCb online Data Acquisition (DAQ) system, which is to a large extend made up of commodity equipment. Results gathered show the Ethernet Controller (Network Interface Card, NIC) driver currently is the major bottleneck, preventing the system from reaching maximal Gigabit Ethernet performance. Receive Descriptor Recycling is implemented under Linux for Intel's e1000 NIC driver, and is shown to successfully remedy the driver inefficiency.**

***Index Terms*— Gigabit Ethernet, Linux networking, NAPI, PCI-X, e1000.**

I. Introduction

BROAD-BAND TECHNOLOGIES enable considerable amounts of data to be transferred through a large network at comfortable speeds. Many modern DAQ systems, such as the LHCb Online DAQ system [1], can therefore depend on commodity hardware to reduce cost and simplify administration. Gigabit Ethernet is particularly attractive because this technology matured over the years and it has a very appealing performance to cost of ownership ratio.

Apart from full load link traffic, of which the effects have been studied in [2] [3], small packets have become more and more important in many real time applications. An example of a popular application, that causes this kind of network traffic, is voice-over-IP (VoIP). For each VoIP connection, 50 network packets are sent every second with a data payload size of 160 bytes or even less [4]. All Linux based firewalls, routers and web caches, where a large amount of small network packets is traveling through, can benefit from the results of this paper.

High reliability of small-sized transmissions is also crucial for the correctness of calibration runs for the LHCb [5] experiment, at the CERN LHC accelerator [6]. Here, small size packets are combined with extreme high packet rates and a near real time constraint. Some work on small packet traffic effects was conducted in [7], but no thorough low-level analysis has yet been performed. Also, Intel has released a document on small packet traffic performance [8], however, it presents merely recommendations which are not substantiated by solid measurements to proof any performance gain.

This paper studies the performance of an Ethernet device with small packets and remedies the associated performance drop by implementing Receive Descriptor Recycling (RDR) for Intel's e1000 NIC Linux driver. First, Section II will elaborate on the LHCb DAQ system, where the low-level measurements are performed. Section III describes the hardware setup and equipment used, followed by an overview of how networking is implemented for Linux in Section IV. Next, Section V and VI summarise the results of the performance measurements and low-level analysis, respectively. Both gather proof implying the Ethernet Controller's driver as the actual bottleneck. Finally, a solution to this bottleneck is provided by the RDR mechanism described in Section VII. Section VIII concludes the paper with a discussion on further work.

II. LHCb DAQ Online System

The LHCb Data Acquisition network [1] relies on Gigabit Ethernet over copper UTP to interconnect custom made electronics with the computing farm of commodity PCs. The farm of processors is partitioned into sub-farms, each one being interfaced to the readout network switch by a gateway (called an Sub-Farm Controller, SFC). A schematic overview of the system is shown in Fig. 1.

The data sources are formed by the front-end electronics, which connect with the switched network through NICs. Different trigger levels [9] allow the initial data rate of 40 MHz to be reduced by two orders of magnitude before entering one of the SFCs. A SFC receives all fragments of a subset of a selected event and assembles them in complete event structures. Via another Gigabit Ethernet switch, it distributes the events to a farm-node that will perform final event filtering to bring the rate to permanent storage down to approximately 200 Hz.

Performance of a SFC is critical for sizing whole the system. It can be measured by means of packet rate, packet loss, response time and throughput. Also, efficient CPU utilisation and resource requirements are important criteria. When optimising for small packet traffic, inevitable trade-offs are encountered, e.g. CPU and resource requirements increase in order to accommodate high rates of small packets, along with a rise in host bus utilisation. Note that transmit performance is not affected by small packet traffic to the same extent as receive performance. This asymmetry exists because the local host cannot usually overwhelm the Ethernet Controller with outgoing traffic.

III. Hardware Setup

All benchmarks were performed on a SFC candidate system running two 32-bit Intel Xeon processors. The DAQ LAN is a Gigabit Ethernet network, connected by cat 5e copper UTP wires. The setup contained two servers and one network processor [10] as a very powerful traffic generator. All servers had hyper-threading [11] enabled.

Both Intel Pro/1000 MT dual and quad port Ethernet NICs were used. On the SRV06 host, version 6.0.54-k2-NAPI of the e1000 network driver was used, on the SFC04 host this was

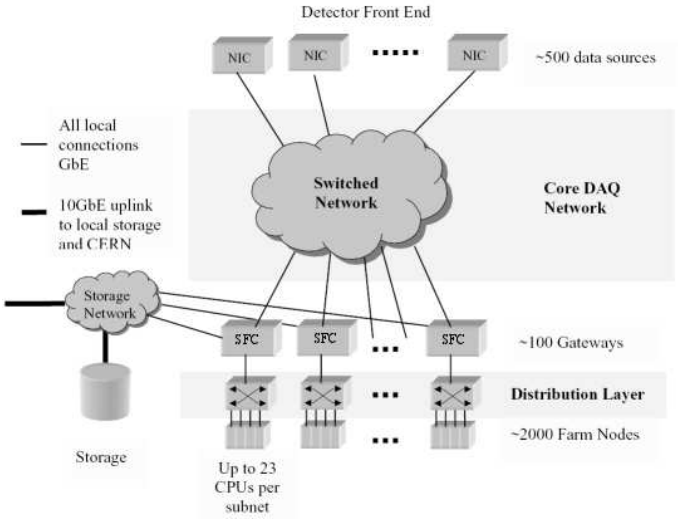


Fig. 1. Schematic view of the LHCb DAQ.

version 5.6.10.1-k2-NAPI. An overview of relevant system specifications is presented in Table I, including the network processor used for frame generation.

TABLE I
SPECIFICATIONS OF HARDWARE USED.

Host	Chipset PCI-X bus	CPU Systembus	Linux Kernel
SRV06	ServerWorks GC 64bit 133MHz	Dual Xeon 2.4GHz 400MHz	Scientific 3.0.4 2.6.12-smp
SFC06	Dell SC1425 64bit 133MHz	Dual Xeon 2.8GHz 800MHz	Scientific 3.0.4 2.6.11-smp

Type	Ports	Chipset	Host bus (max)
Intel Pro/1000MT	dual	82546EB	PCI-X 64b 133MHz
Intel Pro/1000MT	quad	82546EB	PCI-X 64b 133MHz
IBM PowerNP NP4GS3	tri	BCM5700	PCI

IV. LINUX NETWORKING

The tests presented in this paper were performed using raw Ethernet frames using the IEEE 802.3 MAC format [12]; they consist of the 14 byte header followed by variable length user data and a 4 byte CRC-checksum at the end, resulting in a total data overhead of 18 bytes per frame.

When a packet is received, the NIC copies it in the host memory through Direct Memory Access (DMA) and then raises an interrupt. Each device driver maintains a DMA ring (circular buffer) to this end. In order to keep kernel response time small, system interrupts must be disabled for as little time as possible. Softirqs [13] allow the system to schedule deferrable tasks, i.e. Interrupt Service Routine (ISR) tasks that can safely be ran without disabling interrupts. When a NIC raises an interrupt, critical tasks, during which interrupts are disabled, are performed and a softirq for the deferrable tasks is scheduled. The softirq code is executed when the interrupt handler finishes and interrupts have been enabled.

When a system is flooded by packets, a torrent of interrupts is generated by the NIC, which in turn prevents the kernel from processing the packets, since it is spending all its time

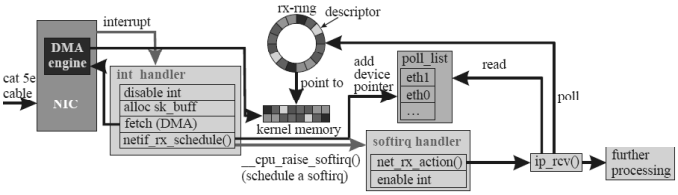


Fig. 2. The NAPI mechanism in Linux kernel 2.6.

in interrupt handling. This is commonly known as a livelock. NAPI [14] applies adaptive interrupt coalescing to offer a compromise between an interrupt driven scheme and a polling scheme, effectively reducing the number of interrupts when flooded. An overview of NAPI is shown in Fig. 2. A NAPI-enabled NIC driver will only interrupt the system on the arrival of the first packet in a batch. Subsequently, it registers to the system that it has work and turns off interrupts concerning receiving packets or running out of receive buffers (referred to as Receive Descriptors, [15]) in its DMA ring. Furthermore, any packets arriving after the DMA ring is filled will be dropped without disturbing the system. This approach meets the goal of preventing a system livelock. Softirq is utilised by NAPI to schedule its polling routine which interrogates devices that registered to offer packets.

The situation so far allows the kernel to process incoming packets as fast as it can. However, since softirqs are invoked upon return from an interrupt handler, the kernel will just try to keep up with packet processing and user level code will never get any CPU time.

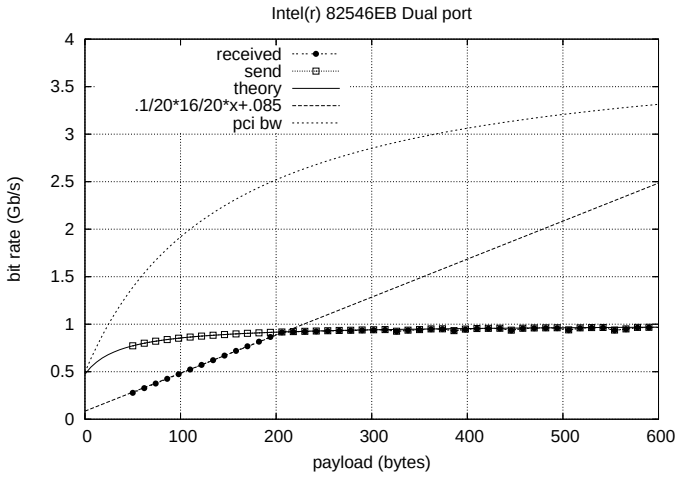
To solve this situation, Linux has implemented a low priority softirq daemon (ksoftirq [13]) which basically checks for any pending softirqs, only if there is free CPU time. This way, user processes are guaranteed not to suffer under heavy traffic. On the other hand, if the system acts as a network node, as is the case in the LHCb DAQ Project, then ksoftirqd will virtually have whole the CPU at its disposal.

From a system performance point of view, sending of packets is much simpler. The NIC driver provides a function through consistent DMA mapping which instructs the hardware to begin transmission. Concurrent access to this function is prevented by use of a spinlock. Under heavy load, when the hardware is unable to keep up, packet transmission may be deferred to a low priority ksoftirqd thread.

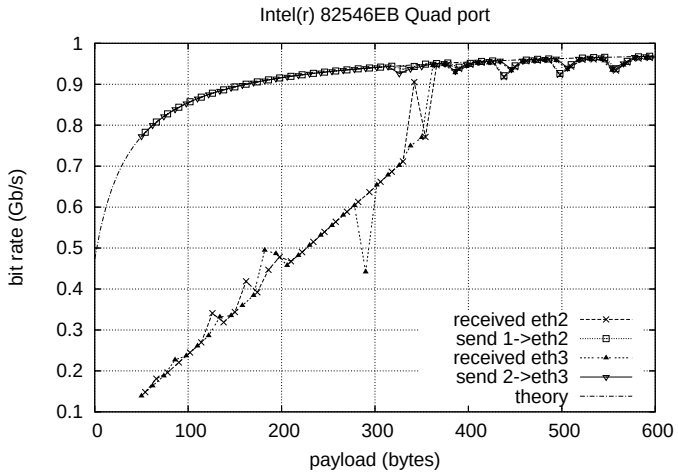
V. PERFORMANCE MEASUREMENTS

A. Receive Throughput

To measure receive throughput, the network processor (NP) acted as a source, flooding the SFC with packets. Fig. 3(a) shows the bit rate measured when receiving on one port of the SFC from one port of the NP with a variable frame size. The bit rate measured at the output of the NP well matches the theory, apart from regular drops. According to the NP documentation, this behaviour is to be expected when pushing this device to the Gigabit Ethernet limit. The SFC receive rate drops accordingly of course. In the range of 0 to 200, the bit rate measured at the input of the SFC is much lower (data is lost). In higher ranges, the sent bit rate is



(a) Dual card, one port, plotted with theoretical Gigabit Ethernet limit, PCI bandwidth and slope.



(b) Quad card, two ports simultaneously, plotted with theoretical Gigabit Ethernet limit.

Fig. 3. Receive throughput from NP (port x) to SFC (eth y) for Intel NIC.

reached (no losses). Receiving on both ports of the dual card simultaneously, produces an identical plot.

Fig. 3(a) also shows the theoretical limit of the PCI bus. It is established that, in the case of operating one single port, this PCI bandwidth will not pose any limit on Gigabit Ethernet transfers, even taking into account the fact that this is a theoretical value which will never actually be reached due to numerous non-idealities. The same figure also shows a linear fit of the first part of the curve. The slope is 0.004, which is in agreement with measurements in [7]. In the continuation of this paper, however, a different conclusion with regard to what contributes to this slope will be reached.

The same measurements were done on an Intel quad port NIC, the resulting plot is shown in Fig. 3(b). The slope of the linear part now amounts to 0.002, a value insensitive to the inter-frame delay setting of the NP. This slope is clearly

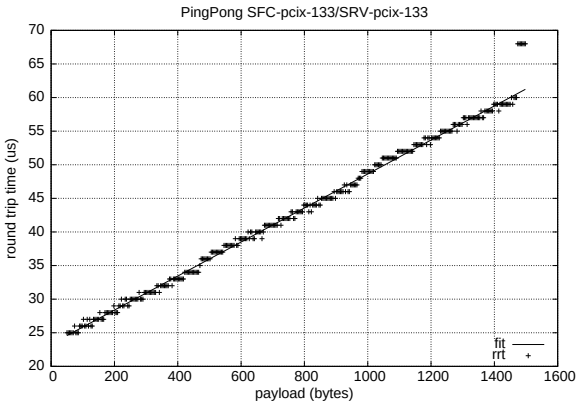


Fig. 4. PingPong benchmark from SFC to SRV, with both Intel dual cards on PCI-X bus 64bit 133MHz. Notice the sudden increase when the MTU is reached (i.e. due to the extra fragmentation overhead). These points were not included in the linear fit.

worse than the 0.004 slope of the dual card. A more careful inspection of the board layout reveals that, whereas the dual port is basically one single Intel FW82546EB chip, the quad card has -naturally- two such chips, but also a lot of supporting chips such as a SST 39VF020 2Mbit Flash memory bank, a Tundra Tsi310 PCI-X bridge and an Altera MAX 7000A PLD. This may give cause to the quad card chipset being not as 'streamlined' as the single-chip dual card. During this test, no PCI-X transfer bottleneck was observed.

B. Response Time

The aim of this test is to retrieve the slope of the delay as function of payload. The minimum Round Trip Time (RTT) was measured for gradually increased payload size, each time taking 1 000 measurements. Note that the effective payload is 46 bytes larger now, since the final packets also contain an ICMP header (8 bytes) and IP header (20 bytes), next to the Ethernet header (14 bytes) and CRC-checksum (4 bytes).

Following [3], comparing this slope with the expected, calculable value gives an insight in the total amount of time spent in different parts of the system.

TABLE II

MEASURED SLOPES OF DIFFERENT SETUPS.

Ping PCI-X 64b	Network	Pong PCI-X 64b	Min. RTT slope (μs/byte)	Difference (μs/byte)
66MHz	1 Gbps	133MHz	0.02739	5.4e-3
100MHz	1 Gbps	133MHz	0.02575	5.7e-3
133MHz	1 Gbps	133MHz	0.02531	5.5e-3

The measured values are presented in Table II, along with their difference compared to the total expected RTT going from SFC04 to SRV06 and back, effectively passing through every component twice. All values are well in agreement with the expected ones [3], given the rather large margin of error (around 20%) for this kind of test.

Apart from the slope, also the intersect was calculated by the fit. All tests had a small intersect of about 23 to 24 μs, confirming that interrupt throttling [16] was disabled on the Ethernet Controller.

VI. Low Level Measurements

The above-mentioned benchmarks measure the real performance of the network, but generally do not allow for identification of bottlenecks present. Therefore, lower level tests as PCI-X traffic analysis and kernel profiling were subsequently carried out, emphasising the hardware and software, respectively.

A. PCI/PCI-X Analysis

Similar to network overhead issues, PCI/PCI-X bus protocols also impose penalties on small packet traffic. In addition to the actual data transfer, each bus transaction requires extra control cycles which introduce overhead (e.g., arbitration latency, address phases, attribute phases, wait states). Bus overhead costs become more pronounced with small packet traffic as bus control and wait cycles consume a larger fraction of each bus transaction.

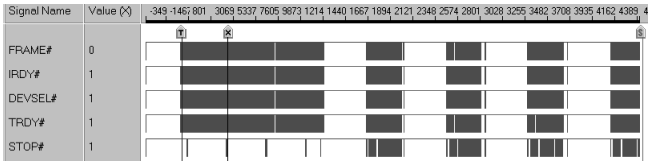


Fig. 5. Transition between smooth and stumble PCI traffic. (1 of 2 ports receiving from NP @ PCI 64bit 66MHz)

Fig. 5 shows the two regimes of transmission across the PCI bus when flooding the NIC with a NP. The dark areas indicate that the signal varies a lot, i.e. frames are transmitted across the bus. Notwithstanding that this plot was taken in PCI mode, exactly the same behaviour was observed when analysing the PCI-X traffic. This section summarises the flow of traffic on the bus, going from the smooth to the stumble regime.

During the smooth regime, all frames are nicely put on the PCI bus behind each other (inter-frame delay of 54 CLKs or 0.8 μs). The Intel Pro/1000 MT Ethernet Controller uses receive and transmit descriptors to keep the books ([15]). Such a descriptor is basically a pointer to memory space, indicating a block of configurable size for the NIC to store a received frame or read in a frame to send. From here on, we will concentrate on reception of frames.

Receive Descriptors (RD) are made available to the NIC in groups of 16 consecutive descriptors. The tail address of this group is provided by a Receive Descriptor Tail (RDT). When a frame is received, the Ethernet Controller uses the next RD in the current RDT to find out where to store the frame. After the DMA transfer finishes, this RD is sent back to the driver to advertise the presence of a frame at that location (delay of 7 CLKs).

Fig. 6 visualises an interpreted version of Fig. 5. It shows left to right and top to bottom the advancing time axis (in clock ticks). The series of black dots are frames that are written to main memory. The following traffic pattern can be extracted for the smooth regime (0 to 14 500 CLKs):

1) Four frames (black dots) are transferred to main memory through DMA bursts with an inter-frame delay of about 45 CLKs or 0.7 μs.

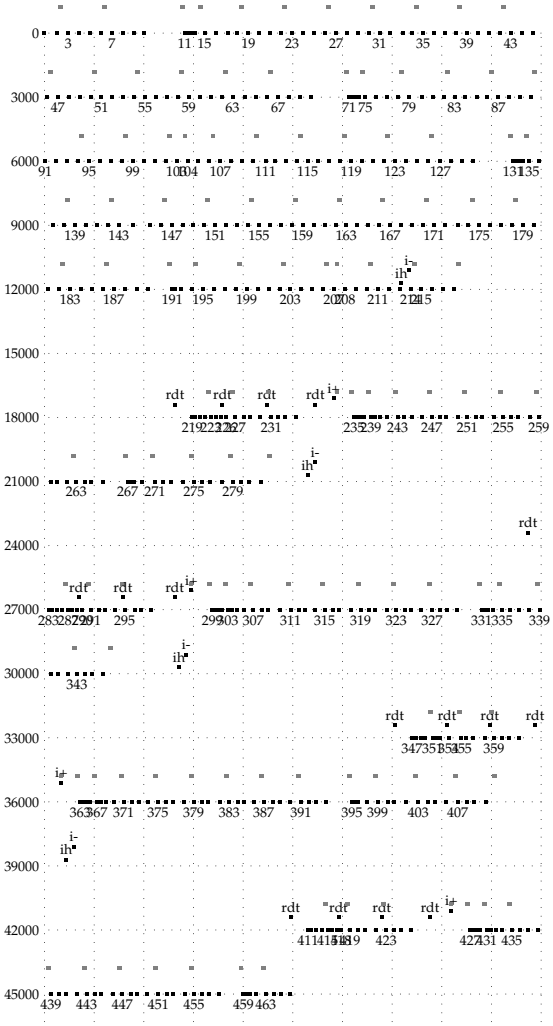


Fig. 6. Trace for the transmission from NP to SRV06 (from Fig. 5).

2) Following afterwards, i.e. 25 CLKs or 0.4 μs, the 4 corresponding RDs are written back to the driver (one grey spot).
3) Step 1 and 2 are repeated 16 times (containing a total of 64 frames), after which a new RDT is employed by the NIC, allowing to fetch 16 new RDs (small gaps in between series of 64 frames).

Under normal circumstances, the driver will provide the NIC with new RDTs (marked 'rdt' in Fig. 6), allowing it to fetch newly allocated RDs in time. This transfer of new RDs continues until all allocated descriptors for that batch have been provided, or, the card's frame receive buffer is nearly full and the card terminates the descriptor transfer by raising the `STOP#` signal in order not to lose any frames. Since a NP is flooding the card during this test (with approximately 1 million packets per second), the latter will happen more and more, as seen on the `STOP#` signal line (Fig. 5). After the descriptor transfer, the frame receive buffer is quickly emptied to regain the time lost during this transfer. All frames are now put on the PCI bus with a minimum delay of 7 CLKs in between.

Clearly, under persistent heavy traffic load, the card will become unable to fetch enough descriptors to keep up with the

high frame rate, and at the same time sent every single frame to the system. The card will have to allow for a FIFO receive buffer overrun and silently drop frames until it has some more RDs to quickly empty the buffer and try start receiving again. The relief is, however, of short duration, as the card will soon have exhausted it (already) very limited pool of RDs, and it will have to wait again.

This is what happens in the second so-called stumble regime (14 500 CLKs to end), where huge gaps of 4 000 CLKs show the lack of RDs is preventing any further traffic until new RDTs are received. This results in many FIFO buffer overruns and a huge packet loss of up to 500k packets per second. Furthermore, the omnipresent `STOP#` indicates that any transfer that takes longer than absolutely necessary is abruptly terminated by the NIC.

In the smooth part, the system can process 1 frame per microsecond (i.e. what the NP sends). When tumbling into stumble regime, this number degrades to a value of merely 0.3 frames per microsecond or 0.28 Gb/s, cfr. earlier results, e.g. Fig. 3(a). This also confirms that the linear part for small packet sizes is caused by this stumble behaviour.

Where one might be tempted to think the PCI bus or NIC is the cause for this bottleneck, calculations on PCI bus utilisation proved otherwise. During the whole trace, peak data rates did not reach any higher than 250 MB/s. Since the practical limit for a PCI bus is about 50% of the theoretical maximum, which is 533 MB/s in this case, this is acceptable and it was already a clear indication that nor the PCI bus, nor the card were responsible for this stumble behaviour. Note that this trace was taken for a single port receiving frames. It is clear that a PCI 64bit 66MHz *will* pose a bottleneck when more than 1 port is operating on the same bus.

Compared to PCI, it was observed that PCI-X traffic contained less errors and less accompanying WAIT states. Along with higher clock frequencies, the more intelligent use of WAIT states and so-called split transactions instead of delayed transactions are among the most important features of the PCI-X protocol.

A look at the kernel's memory management information, provided by the `/var/slabinfo` file, made clear that memory access was responsible for the large gaps in stumble regime. These gaps can be explained by the DMA slab freeings. Normally, a CPU keeps a freed DMA slab in its bucket (one bucket per CPU). Occasionally, however, a large quantity of main memory is reallocated by the driver and then the main memory needs to be accessed, preventing others, e.g. DMA accesses by the NIC, from accessing the main memory. Beware, not the memory bank technology but the way the driver seems to handle memory access is causing the bottleneck. Simple on the back-of-the-envelope calculations show that the DDR-2 400MHz (PC2-3200) provides enough raw bandwidth.

It is clear that the Intel Pro/1000 MT Ethernet Controller card, when used in combination with a fast PCI-X host bus, will not become a bottleneck, even for quad port operation. Therefore, the next section takes a closer look to the software side of the network infrastructure, i.e. the Linux operating system and the e1000 Ethernet Controller driver.

B. Kernel Profiling

This last test employed the `OProfile` package [17], a system-wide profiler for Linux systems, capable of profiling all running code at low overhead. It reports on the number of times a certain function (in user space, kernel, driver,...) has been called and what percentage of the total samples taken this represents.

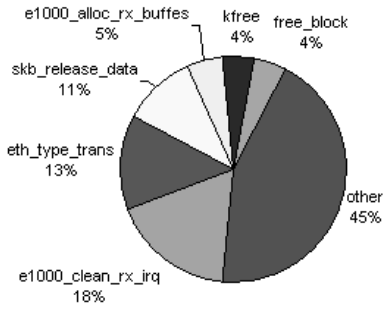


Fig. 7. Summary of relative occupation of the CPU during a 5 minute flood by the NP (1 of 2 ports @ PCI-X 64bit 133MHz).

Results of the `OProfile` test are summarised in Fig. 7. Analysing the exact content of the most frequently made calls, it is established that all are involve with freeing and reallocating RDs. This gave cause for the idea to tune the e1000 driver to more cleverly handle this RD processing, and implement some kind of Receive Descriptor *recycling*. This idea, along with its implementation in the e1000 source code, will be covered in the next section.

VII. Receive Descriptor Recycling

The `e1000_main.c` file of the e1000 driver source code [18] contains most of the driver's basic functionality. Analysing this source for packet reception handling on driver level, points out that the most frequently called functions of Fig. 7 are all related to one single very time consuming operation: the freeing of large heaps of memory. Especially when talking small payload sizes, it is clear that the situation only worsens due to more descriptors and thus more memory management overhead.

It is this overhead that prevents the driver from quickly sending new RDs to the Ethernet Controller, as they need to be freed and reallocated first. This is why the *Receive Descriptor Recycling* mechanism was implemented. The idea is to allocate a fixed number of permanent descriptors which are *reused* every time, effectively taking away the need for the costly reallocation overhead. The only processing that remains to be done is resetting some fields in the descriptors. The remainder of this section will outline the details of this implementation.

First it was needed to store the permanent buffers in the `e1000_adapter` struct (see `e1000.h`), so they became associated with each Ethernet Controller present in the system. For this, it was extended by two members: an

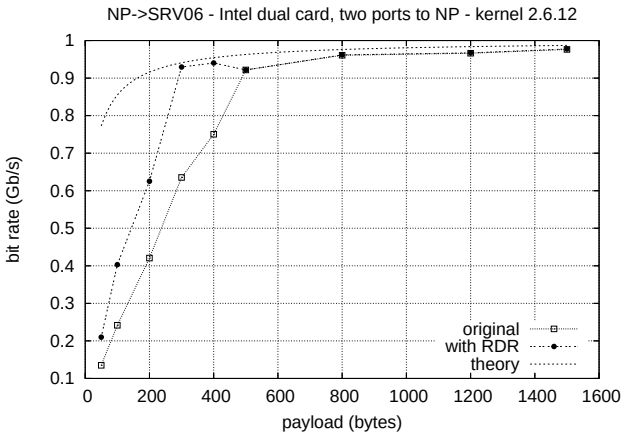


Fig. 8. Receive throughput from NP to SRV for *two* ports on a dual Intel NIC using official and patched driver.

array to store the pre-allocated socket buffers, and an array to point to the associated data fields. A fixed array size of 2 048 was chosen, which was the `RxDescriptors` e1000 driver parameter used in all previous tests.

During the driver initialisation, the array is allocated in memory and the data pointers are set up. To prevent the driver from freeing the allocated array, the number of users for each socket buffer is artificially increased to 2 by calling `skb_get()`. As long as the number of users remains higher than 2, `kfree_skb()` will never free them, since it believes someone is still using the socket buffer and its data.

This is realised by altering the driver function call flow in `e1000_alloc_rx_buffers()`. Here the call to `dev_alloc_skb()`, which would return a fresh allocated socket buffer (skb), is replaced by returning one of our pre-allocated skbs in the array. Next, the skb reset is implemented by the newly added `reset_skb()` function.

Please refer to [19] for a detailed overview of exact code implementations.

To check for any performance increase for small packet sizes, the 'Receive Throughput' test (see Section V-A) was repeated. For small frame sizes, the RDR-enabled driver was able to reduce packet loss by 40%, see Fig. 8. The influence is the most clear for short delay packets. The performance of higher payloads remained unchanged with RDR.

VIII. Conclusions and Further Work

Several benchmarks were performed on Gigabit Ethernet hardware and the Ethernet Controller driver. Among the aspects analysed in this work were throughput, packet loss, response time and efficient resource utilisation. Emphasis was put on small packet size network traffic, to be used for calibration of the LHCb experiment. It was shown that the current bottleneck lies in the way the e1000 driver handles the Receive Descriptor memory management, which proofs to be fatal for small packet sizes. In order to remedy the situation, a Receive Descriptor Recycling mechanism was proposed and implemented in the official e1000 driver. Results have shown an improvement of small packet size performance by 40% in terms of increase in throughput and reduction of packet loss.

The knowledge gathered by the reported measurements, will be used to optimise the design of the DAQ infrastructure. The proposed Receive Descriptor Recycling will be tested in the real-life DAQ Prototype Farm environment. Further work also includes enhancing the RDR performance by implementing the pre-allocated buffers in such a way that they do not cross cache lines as to prevent numerous cache trashing.

IX. Acknowledgements

This work has been funded by the Summer Student Internship Committe of CERN, Geneva, within the framework of the Summer Student Programme 2005.

References

[1] A. Barczyk, J.-P. Dufey, B. Jost, and N. Neufeld, "The new LHCb trigger and DAQ strategy: A system architecture based on gigabit-ethernet," *IEEE Trans. Nucl. Sci.*, vol. 51, pp. 456–460, June 2004.

[2] A. Barczyk, A. Carbone, J.-P. Dufey, D. Galli, B. Jost, U. Marconi, N. Neufeld, G. Peco, and V. Vagnoni, "Reliability of datagram transmission on gigabit ethernet at full link load," CERN, Geneva, CERN/LHCb 2004-030, 2004.

[3] R. Hughes-Jones, P. Clarke, and S. Dallison, "Performance of 1 and 10 gigabit ethernet cards with server quality motherboards," *Future Generation Computer Systems*, vol. 21, pp. 469–488, Apr. 2005.

[4] Voice Over IP - per call bandwidth consumption. [Online]. Available: http://cisco.com/warp/public/788/pkt-voice-general/bwidth_consume.html

[5] LHCb, "LHCb technical proposal," CERN, Geneva, CERN/LHCC 98–4, 1998.

[6] O. S. Bruning, P. Collier, *et al.*, Eds., *LHC Design Report*. Geneva, CH: CERN, 2004.

[7] A. Barczyk, D. Bortolotti, A. Carbone, J.-P. Dufey, D. Galli, B. Gaidioz, D. Gregori, B. Jost, U. Marconi, N. Neufeld, G. Peco, and V. Vagnoni, "High rate packets transmission on ethernet LAN using commodity hardware," *IEEE Trans. Nucl. Sci.*, accepted for publication.

[8] "Small packet traffic performance optimization for 8255x and 8254x ethernet controllers," Application Note AP-453 rev. 1.0, Intel, Sept. 2003.

[9] LHCb, "LHCb trigger system technical design report," CERN, Geneva, Tech. Rep. TDR 10, Sept. 2003.

[10] J. R. Allen, Jr., *et al.*, "IBM PowerNP network processor: Hardware, software, and applications," *IBM J. Res. and Dev.*, vol. 47, Mar. 2003.

[11] "Hyper-threading technology on the Intel Xeon processor family for servers," White Paper, Intel, Oct. 2002.

[12] *Ethernet LAN Medium Access Control (MAC) Specification*, IEEE Std. 802.3, 1985.

[13] D. Bovet, *Understanding the Linux Kernel*, 2nd ed. O'Reilly, 2003.

[14] J. Salim, R. Olsson, *et al.*, "Beyond softnet," in *Proc. 5th Ann. Linux Showcase and Conf.*, Oakland, California, Nov. 2001.

[15] "PCI/PCI-X family of gigabit ethernet controllers software developer's manual," rev. 2.5, Intel, July 2005.

[16] "Interrupt moderation using Intel gigabit ethernet controllers," Application Note AP-450 rev. 1.1, Intel, Sept. 2003.

[17] OProfile development pages. [Online]. Available: http://oprofile.sourceforge.net

[18] e1000 source code at the Linux cross reference project. [Online]. Available: http://lxr.linux.no/source/drivers/net/e1000/

[19] C. Walravens and B. Gaidioz, "Low level gigabit ethernet analysis for the LHCb computing farm," CERN, Geneva, CERN/LHCb 2005-091.